Decorator Pattern

The decorator pattern Is a pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class. The decorator pattern is often used to extend the functionality of a class without having to use inheritance.

Property	Decorator Pattern	Inheritance
Adding behavior	Behavior is added to an object dynamically, at run-time.	Behavior is added to a class statically, at compile-time.
Modularity and reusability	Behavior added by a decorator can be easily reused with other objects.	Inherited behavior is fixed at compile-time and cannot be changed easily.
Use cases	Suitable for situations where the behavior of an object needs to be changed frequently, or where a large number of objects with different behavior are needed.	Suitable for situations where the behavior of a class is well-defined and not expected to change frequently.

Sample Code:

```
9 usages  4 implementations
interface Coffee {
    10 usages  4 implementations
    String brew();
}
3 usages
public class SimpleCoffee implements Coffee {
    10 usages
    public String brew(){
        return "Coffee";
    }
```

```
abstract class CoffeeDecorator implements Coffee{
    2 usages
    protected Coffee decoratedCoffee;

    2 usages
    public CoffeeDecorator(Coffee decoratedCoffee){
        this.decoratedCoffee = decoratedCoffee;
    }

    10 usages  2 overrides
    public String brew(){
        return decoratedCoffee.brew();
    }
}
```

```
1 usage
class MilkDecorator extends CoffeeDecorator {
    1 usage
    public MilkDecorator(Coffee decoratedCoffee){
        super(decoratedCoffee);
    }

    10 usages
    public String brew(){
        return super.brew() + " with Milk";
    }
}

2 usages
class SugarDecorator extends  CoffeeDecorator {
    2 usages
    public SugarDecorator(Coffee decoratedCoffee){
        super(decoratedCoffee);
    }

    10 usages
    public String brew(){
        return super.brew() + " with Sugar";
    }
}
```

```
public class Main {
    public static void ma[...]gs) {
        Coffee myCoffee = new SimpleCoffee();
        Coffee myCoffee2 = new SimpleCoffee();
        Coffee myCoffee3 = new SimpleCoffee();

        System.out.println(myCoffee.brew());
        System.out.println(myCoffee2.brew());
        System.out.println(myCoffee3.brew());

        System.out.println();

        myCoffee = new MilkDecorator(myCoffee);
        myCoffee2 = new SugarDecorator((myCoffee2));
        System.out.println(myCoffee.brew());
        System.out.println(myCoffee2.brew());
        System.out.println(myCoffee3.brew());

        myCoffee = new SugarDecorator(myCoffee);
        System.out.println(myCoffee.brew());


    }
}
```

Team Loyal

Members:

Rico Miles Quiblat

John Quinnvic Taboada

Le Bronn Samson